

计算机组成

# 基本认识、数制

高小鹏

北京航空航天大学计算机学院

## 目录

- 计算机的基本认识
- 计算机的应用及分类
- 计算机的基本硬件构成
- 计算机的层次结构



北京航空航天大学计算机学院  
School of Computer Science and Engineering, Beihang University

# 第一台数字式多用途计算机

ENIAC: Electronic Numerical Integrator and Calculator

- 时间: 1946年; 用途: 计算炮弹弹道
- 美国宾夕法尼亚大学

真空管	重量	面积	功耗	性能
18800	30吨	100m <sup>2</sup>	150KW	5000加法/秒

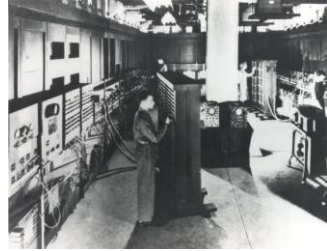
两位主设计师



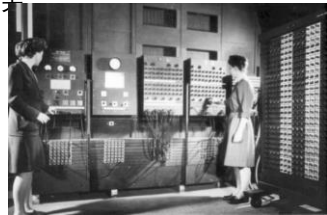
John William Mauchly  
(1907-1980)



J. Presper Eckert  
(1919-1995)



Cpl. Irwin Goldstein设置功能表开关

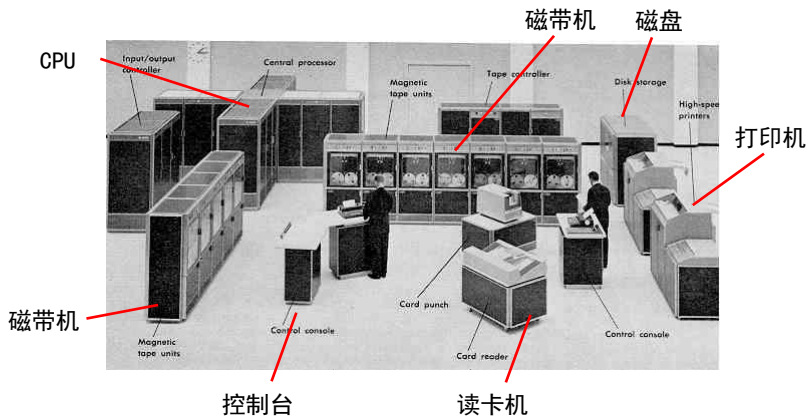


程序员Betty Jean Jennings (左) 与 Fran Bilas (右)

3

# 大型机时代: 上世纪50年代~60年代

核心技术: 计算机



龙头企业	价格	用户	技术
IBM	1M美元	大型企业	COBOL, Fortran, 分时OS

大型机~Mainframe

4

## 小型机时代：上世纪70年代

- 核心技术：集成电路（Integrated circuits）



龙头企业	价格	用户	技术
Digital, HP	10K美元	大学, 实验室	C, UNIX

小型机~Minicomputer

北京航空航天大学计算机学院  
School of Computer Science and Engineering, Beihang University

## PC时代：上世纪80年代~2000中期

- 核心技术：微处理器（Microprocessor）



龙头企业	价格	用户	技术
Apple, IBM	1K美元	个人	Basic, Java, Windows

PC~Personal Computer

北京航空航天大学计算机学院  
School of Computer Science and Engineering, Beihang University

## 后PC时代：2000以后<sup>1/2</sup>

- 核心技术：无线网络(Wireless Network), ARM CPU



Motorola V70  
360度旋盖  
2002



Nokia N-Gage  
全球首款游戏手机  
2003



Apple iPhone  
全面定义智能手机  
2007

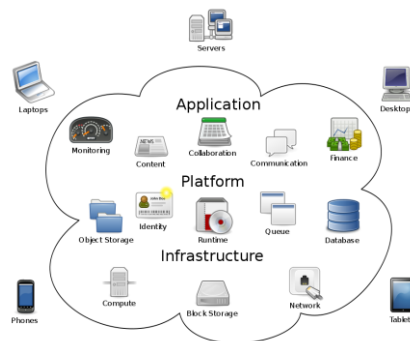
龙头企业	价格	用户	技术
Motorola, Nokia, Apple, Google	500美元	个人	iOS, Android

7

## 后PC时代：2000以后<sup>2/2</sup>

- 核心技术：局域网，宽带互联网

### 云计算



龙头企业	用户
Amazon, Google, 阿里	无法(不愿)负担高端计算与存储设备的用户

8

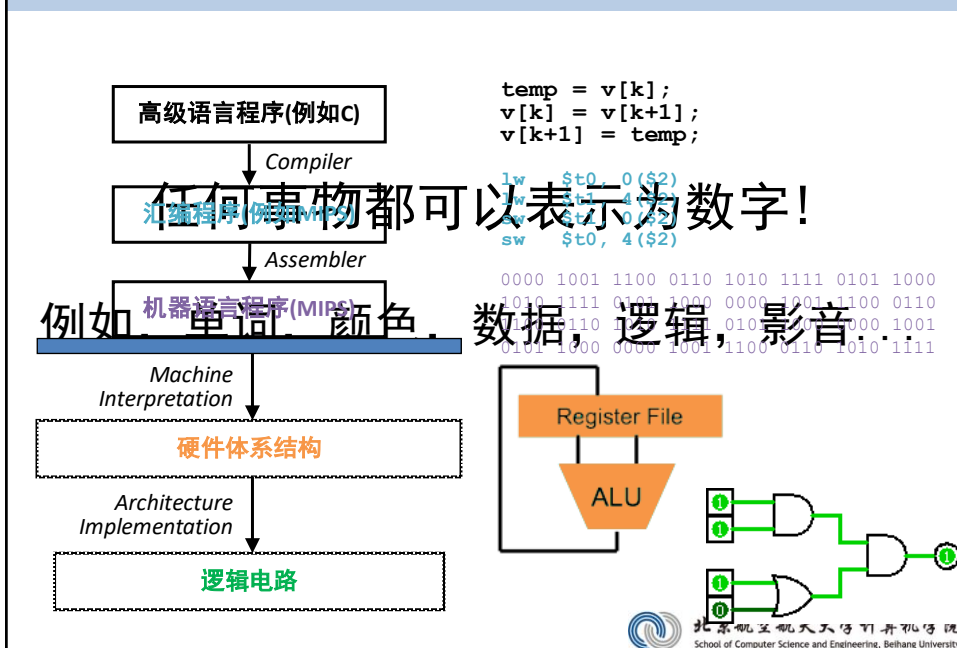
## 重大认识

- 1、层次化表示与解释
- 2、摩尔定律
- 3、局部性与存储层次
- 4、并行性
- 5、性能度量与改进

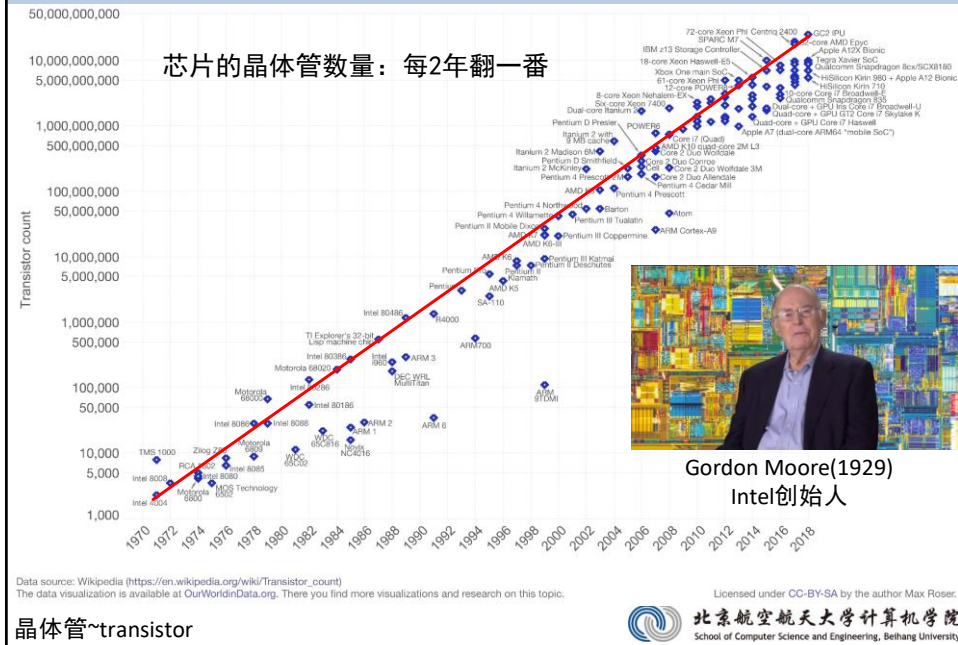


北京航空航天大学计算机学院  
School of Computer Science and Engineering, Beihang University

## 重大认识#1：层次化表示与解释

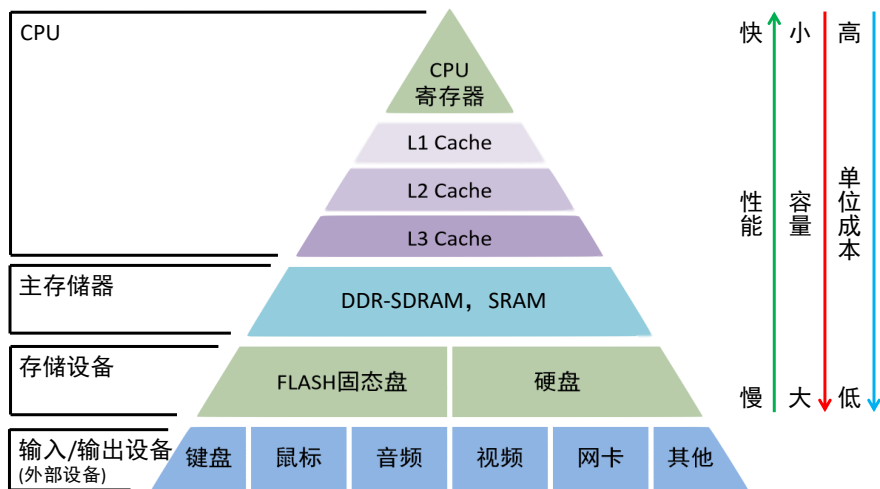


## 重大认识#2：摩尔定律



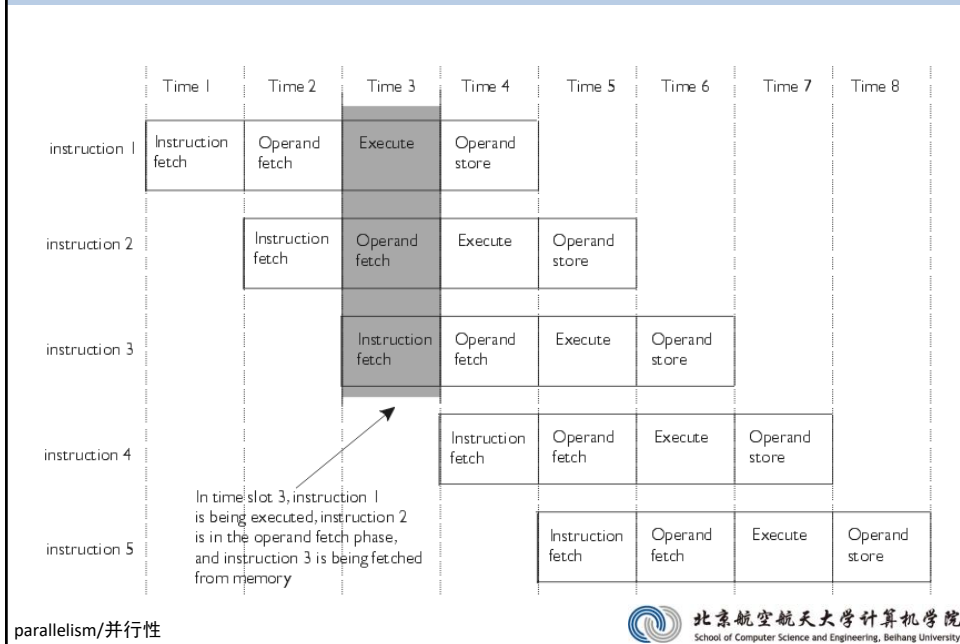
## 重大认识#3：局部性与存储层次

- 局部性：体现了程序内在的固有基本规律
- 存储层次：应用了局部性原理，是性能、容量、成本的设计折中



SRAM~Static Random Access Memory; SDRAM~Synchronous Dynamic RAM  
DDR-SDRAM~Double Data Rate SDRAM

## 重大认识#4：并行性



## 重大认识#5：性能度量与改进

- 需要比较各类体系结构，以及量化性能的改进程度
- 这些都是关于程序完成时间的（被称为延迟）
  - ◆ 由两部分组成：建立时间、执行时间
- 发挥性能的关键在于深入探索
  - ◆ 局部性
  - ◆ 并行性
  - ◆ 专用硬件特性，例如专用指令（如矩阵乘法指令、FFT指令）

latency/延迟 setup/建立 locality/局部性

# Amdahl定律

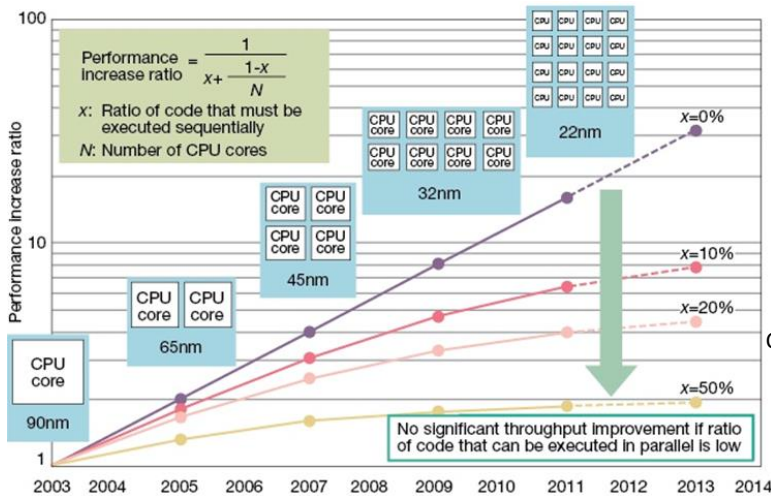


Fig 3 Amdahl's Law an Obstacle to Improved Performance



**Gene Amdahl**  
Computer Pioneer  
Ph.D. Wisconsin  
1952



北京航空航天大学计算机学院  
School of Computer Science and Engineering, Beihang University

## 目录

- 计算机的基本认识
- 计算机的应用及分类
- 计算机的基本硬件构成
- 计算机的层次结构

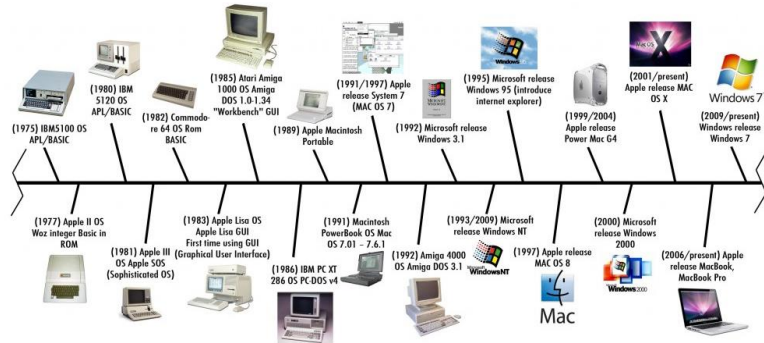


北京航空航天大学计算机学院  
School of Computer Science and Engineering, Beihang University



## 个人计算机

- 定位：是以较小代价为个人用户提供较高的性能
- 范围：最为广泛，如台式机、笔记本电脑都是PC范畴
  - ◆ PC始终是最大的计算机市场之一
- 软件：办公软件、开发软件、娱乐软件、游戏软件等



来源: <https://custommsx.com/history-of-the-personal-computer/>

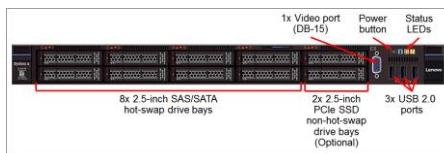
北京航空航天大学计算机学院  
School of Computer Science and Engineering, Beihang University

## 服务器<sup>1/3</sup>

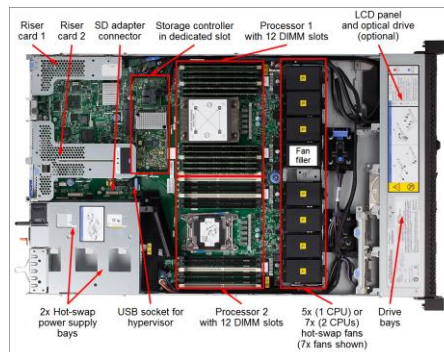
- 定位：承载大负载的任务，例如科学计算、Web访问等
- 软件：Web服务、数据库、科学计算软件、模拟系统等
- 应用：部署在机房中，通过网络对外提供计算与存储服务



外视图



前视图：8个2.5英寸SATA硬盘



内部视图

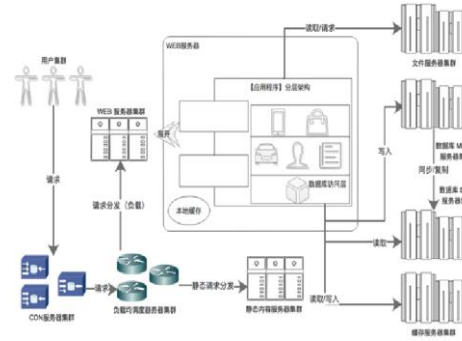
Lenovo System x3550 M5

服务器~Server

北京航空航天大学计算机学院  
School of Computer Science and Engineering, Beihang University

## 服务器<sup>2/3</sup>--服务器集群

- 服务器集群：多台服务器聚合在一起，以分布式对外提供服务
  - 微信、淘宝、新浪等的运行平台

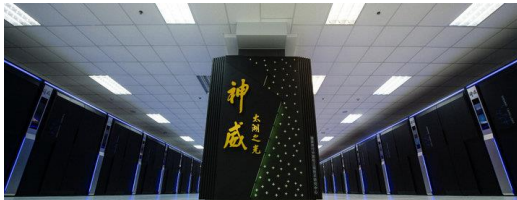


集群~Cluster

北京航空航天大学计算机学院  
School of Computer Science and Engineering, Beihang University

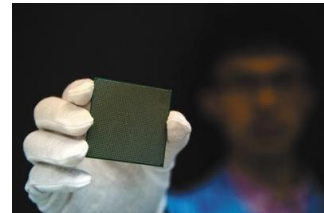
## 服务器<sup>3/3</sup>--超级计算机

- 超级计算机：更大规模的服务器集群，聚合的处理器总量大约在10万颗量级，主要以并行方式提供超级计算能力
  - 用途：如天气预报、地质勘探、核爆模拟、蛋白质结构分析等



CPU数量：申威 26010 x 40960  
计算能力：全球 72 亿人口用计算器不间断计算 32 年  
系统功耗：15.371MW

机柜	超节点 x 4
超节点	运算插件 x 32
运算插件	运算节点 x 4
运算节点	申威 26010 x 2



中国芯：申威 26010

厂商	江南计算技术研究所
指令集	64位 RISC
核心数	260
主频	1.45 GHz
峰值性能	3.06 TFlop/s

超级计算机~Super Computer, SC

北京航空航天大学计算机学院  
School of Computer Science and Engineering, Beihang University

## 嵌入式计算机<sup>1/2</sup>

- 定位：面向特点应用，作为一个子系统被内嵌在整个系统中
- 范围：最为广泛，无所不在
  - ◆ 冰箱、洗衣机、微波炉、数字电视、汽车、高铁、飞机、轮船、卫星、4G网络、WIFI、机器人.....
  - ◆ 最大的计算机市场



北京航空航天大学计算机学院  
School of Computer Science and Engineering, Beihang University

## 嵌入式计算机<sup>2/2</sup>

- 普遍更重视成本、功耗、可靠性
  - ◆ 通常不把性能作为首要指标
- 某些应用中，还非常强调实时特性
  - ◆ 例如视频播放器

实时~Real Time

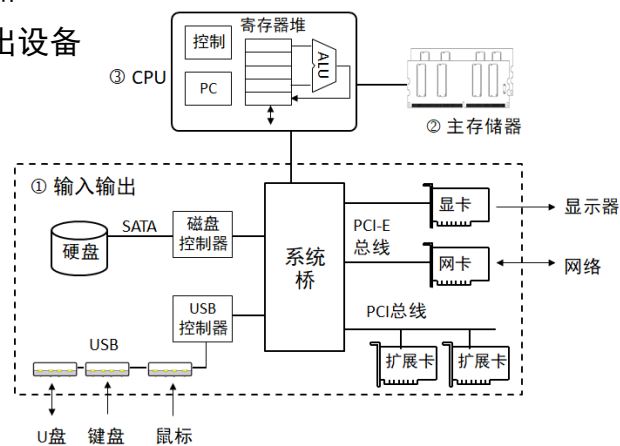
北京航空航天大学计算机学院  
School of Computer Science and Engineering, Beihang University

# 目录

- 计算机的基本认识
- 计算机的应用及分类
- **计算机的基本硬件构成**
- 计算机的层次结构

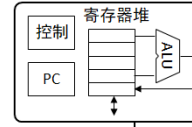
## 计算机的典型硬件组成

- CPU
- 主存储器
- 输入输出设备



CPU<sup>1/2</sup>

- CPU是计算机的最核心部件，其功能是能够执行由一组指令构成的程序
- CPU可以抽象为2大部分：数据通路、控制器
  - ◆ 数据通路：执行指令所需的计算与存储的功能部件集合
    - PC：类似于C语言的指针，指向要执行的指令
    - ALU：完成各类计算，如加减乘除
    - 寄存器堆：临时性存储参与运算的数据以及运算结果
  - ◆ 控制器：根据指令控制数据通路的各个部件完成相应操作
    - 例如加法指令，就需要控制ALU完成加法运算



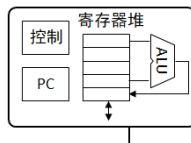
CPU基本模型

程序计数器~Program Counter(PC)  
 算术逻辑单元~Arithmetic and Logic Unit(ALU)

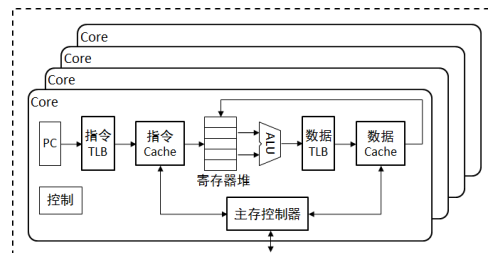
北京航空航天大学计算机学院  
 School of Computer Science and Engineering, Beihang University

CPU<sup>2/2</sup>

- 现代CPU通常包含cache、MMU
- 为了提高性能，现代主流CPU一般为多核结构
  - ◆ 每个核心相当于过去的一个完整CPU



CPU基本模型



现代主流CPU基本结构

高速缓存~cache  
 存储管理单元~Memory Management Unit(MMU)

北京航空航天大学计算机学院  
 School of Computer Science and Engineering, Beihang University

## 主存储器

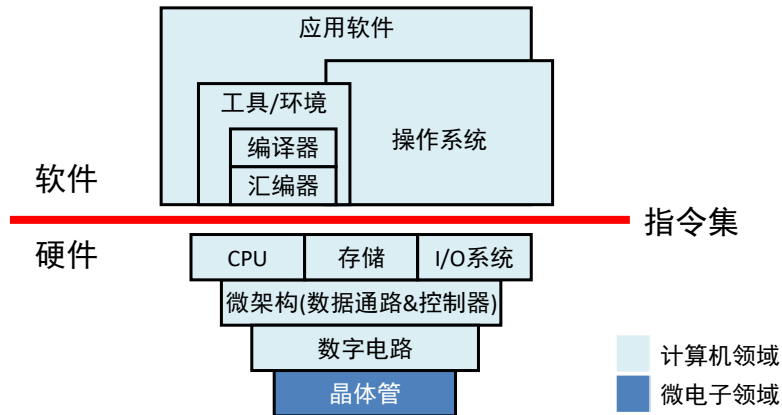
- 主存储器（简称主存）：存储程序以及数据
  - ◆ 程序示例：Excel.exe就是一个程序，平时是存储在硬盘中的；被鼠标双击后，它就会被Windows操作系统加载到主存中
  - ◆ 数据示例：Excel中打开的某个excel文件就是Excel.exe运行所需要的数据
- 主存很多时候也被称为内存
  - ◆ 顾名思义，既然有主存和内存的概念，就有辅存和外存的概念
  - ◆ 辅存或外存通常是指硬盘、光盘等设备
- 主存容量
  - ◆ 早期计算机主存只有数K字节
  - ◆ 现代PC主存容量达到数G字节以上

## 目录

- 计算机的基本认识
- 计算机的应用及分类
- 计算机的基本硬件构成
- 计算机的层次结构

## 软硬件层次结构

- 一个完整的计算机系统，可以划分为硬件和软件两大部分
  - ◆ 其中，两者之间的交互界面就是指令集



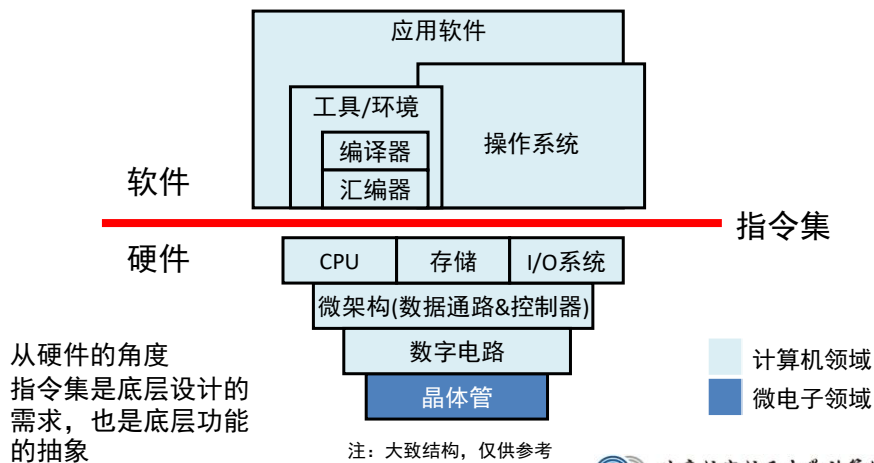
注：大致结构，仅供参考



北京航空航天大学计算机学院  
School of Computer Science and Engineering, Beihang University

## 软硬件层次结构：指令集

- 硬件功能被抽象为一组基础操作，如加法运算、减法运算等
- 这些操作被称指令；指令的集合就是指令集



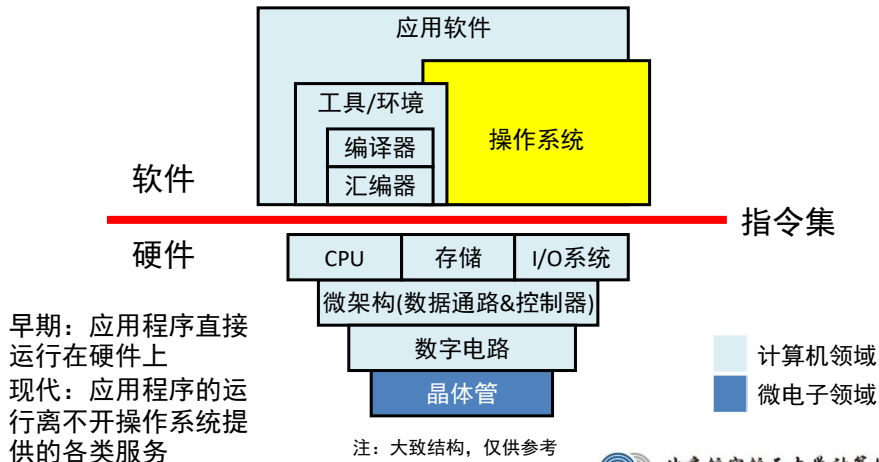
注：大致结构，仅供参考



北京航空航天大学计算机学院  
School of Computer Science and Engineering, Beihang University

## 软硬件层次结构：操作系统

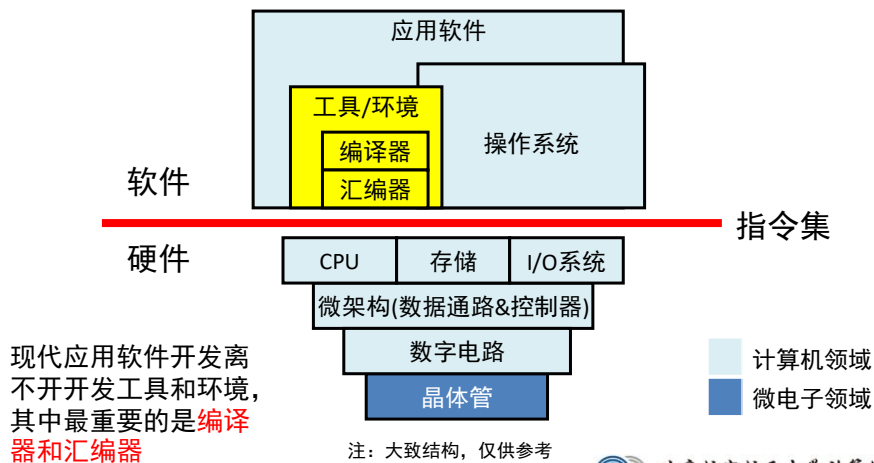
- 操作系统：管理软件和硬件资源，为应用程序提供丰富的功能
  - ◆ 如Windows、Linux、IOS、Andorid



北京航空航天大学计算机学院  
School of Computer Science and Engineering, Beihang University

## 软硬件层次结构：编译器/汇编器

- 编译器：把高级程序设计语言（如C程序）转换为汇编程序
- 汇编器：把汇编程序转换为二进制机器指令

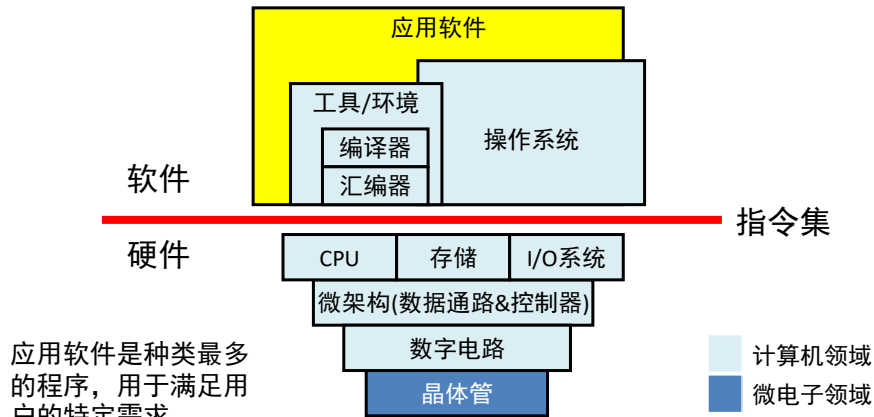


北京航空航天大学计算机学院  
School of Computer Science and Engineering, Beihang University



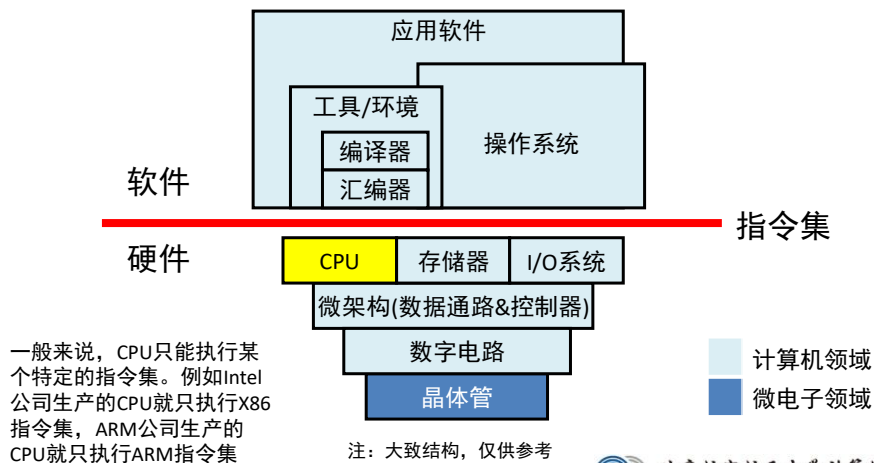
## 软硬件层次结构：应用软件

- 种类最为繁多，用于满足用户的特定需求
  - ◆ ①PC上的各类办公软件、游戏、浏览器等；②服务器上的各类Web服务软件、科学与工程计算软件等；③手机上的微信等



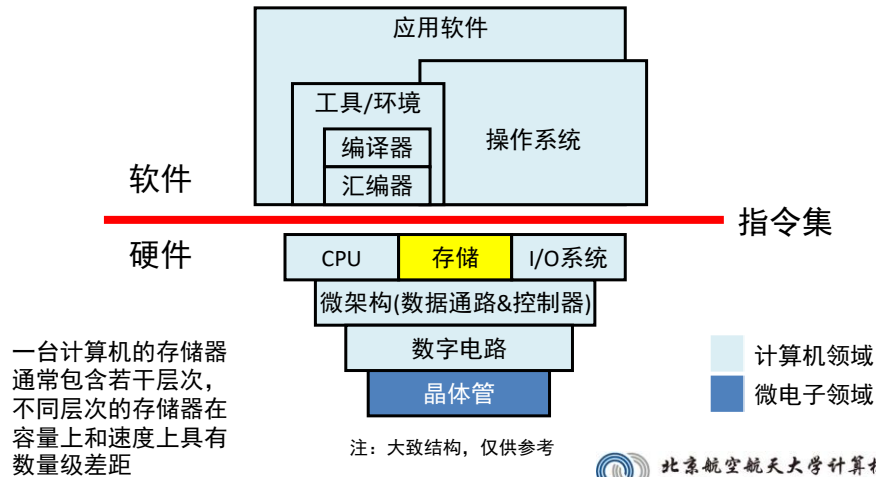
## 软硬件层次结构：CPU

- 计算机硬件中最核心的部件。它执行程序的基本原理：
  - ◆ 从存储器读取一条指令；执行指令功能；计算下条指令在存储器的位置
  - ◆ 不断的重复上述过程



## 软硬件层次结构：存储器

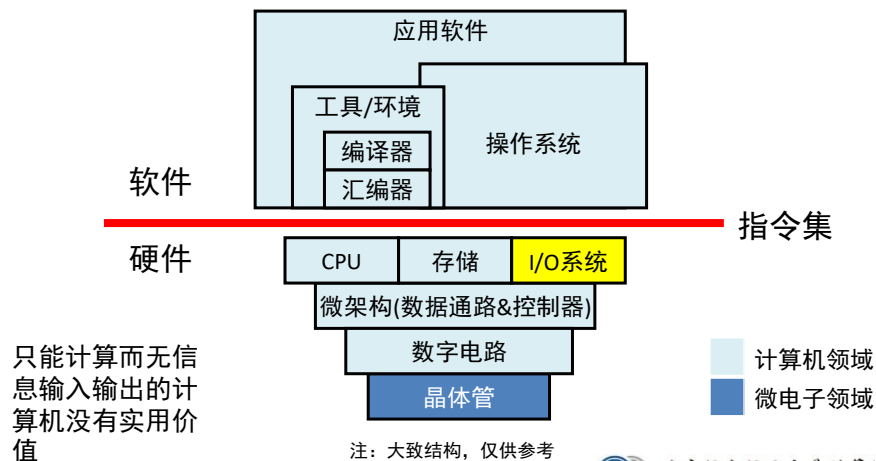
- 存储器是计算机硬件中第2重要的组成部分
- 今天使用的绝大多数计算机应该称为存储程序计算机
  - 构成程序的指令和数据都以二进制数字的形式存储在存储器中



北京航空航天大学计算机学院  
School of Computer Science and Engineering, Beihang University

## 软硬件层次结构：输入/输出系统

- 输入系统负责信息进入计算机，输出系统负责信息从计算机输出
  - 如鼠标、键盘、无线网卡

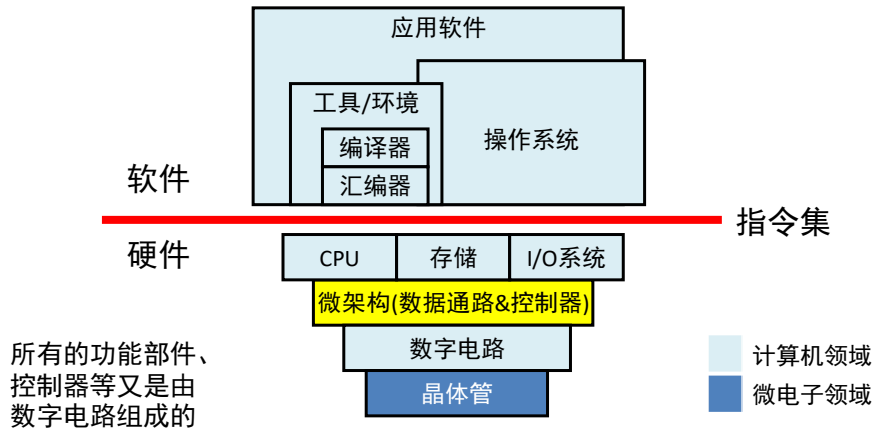


北京航空航天大学计算机学院  
School of Computer Science and Engineering, Beihang University

## 软硬件层次结构：微架构

### □ 芯片的内部结构就是微架构

- ◆ CPU这样的芯片，其内部可以划分为两大部分：数据通路、控制器
- ◆ 其中，数据通路又包含一组功能部件，如寄存器堆、运算单元等



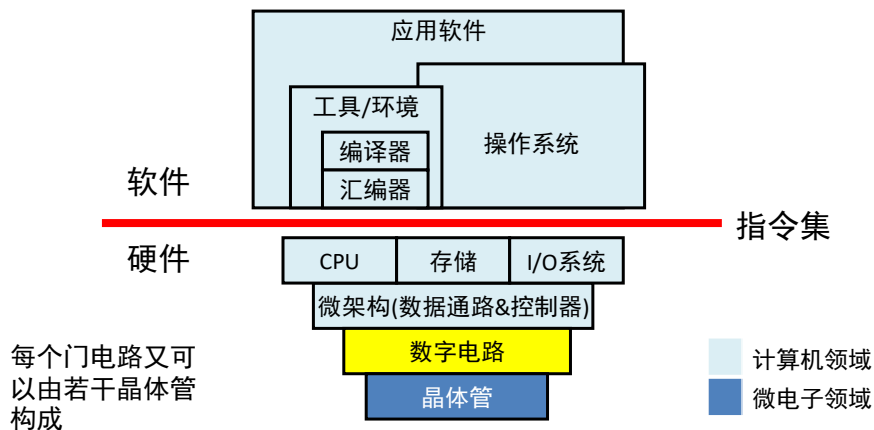
注：大致结构，仅供参考

北京航空航天大学计算机学院  
School of Computer Science and Engineering, Beihang University

## 软硬件层次结构：数字电路

### □ 数字电路包括组合电路和时序电路两大部分

- ◆ 组合电路：具有计算功能的电路，如与门、或门、非门等
- ◆ 时序电路：具有存储功能的电路，如寄存器等

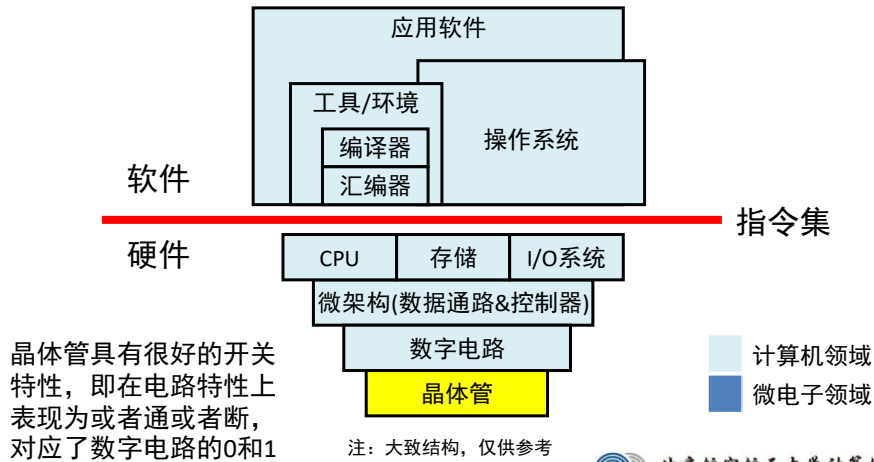


注：大致结构，仅供参考

北京航空航天大学计算机学院  
School of Computer Science and Engineering, Beihang University

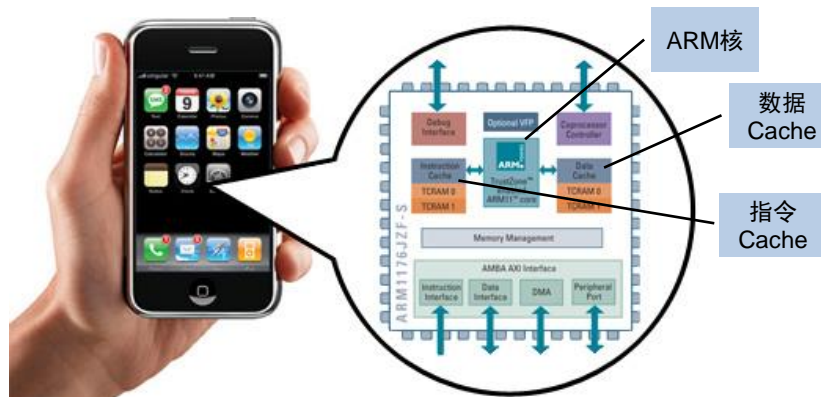
## 软硬件层次结构：晶体管

- 晶体管是实现数字电路的基本单元
  - 晶体管是目前实现开关电路的主要技术手段
  - 晶体管的开关性能和功耗直接影响芯片的频率、功耗等



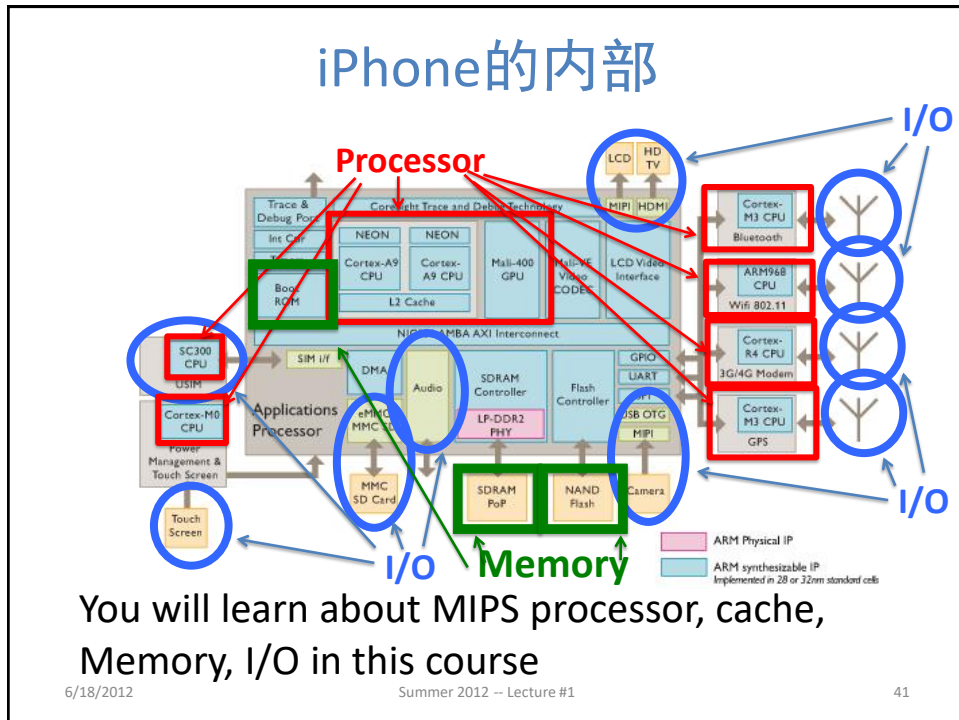
北京航空航天大学计算机学院  
School of Computer Science and Engineering, Beihang University

## Advanced RISC Machine (ARM) instruction set inside the iPhone



You will learn how to design and program a related RISC computer: MIPS

instruction set: 指令集; Cache: 高速缓存



## 为什么讲授MIPS而不是x86

- 搞懂一个CPU，就很容易理解其他的CPU
- x86指令集过于复杂
  - ◆ 属于早期设计，后续发展过程中，为了兼容性而不得不采用打补丁的策略
  - ◆ 设计缺乏系统性，细节过于繁杂
  - ◆ 课程时间有限，我们不能掉入细节的陷阱
- MIPS(以及ARM)强调**效能**，x86则是强调**性能**
  - ◆ 特别是在嵌入式、移动端，效能第一重要
  - ◆ 2010：ARM出货60亿片，x86出货3亿



## 目录

- 常见进制及其转换
- 常见术语
- 二进制加法
- 整数的二进制表示方法
- 浮点数的二进制表示方法
- 补码的几种常见运算

## 数的表示方法

- 在计算机中，任何对象都被表示为一组0/1串
  - ◆ 即使是上面这句话，也是由一组0/1串构成的！
  - ◆ 这就是所谓的二进制表示
- 如何用二进制表示数呢？
  - ◆ 让我们从熟悉的十进制开始介绍

## 十进制数的表示方法

- 示例：10进制的1234，其值的计算过程可以表示为

$$1234_{10} = \overbrace{1 \times 10^3}^{1\text{个}10^3} + \overbrace{2 \times 10^2}^{2\text{个}10^2} + \overbrace{3 \times 10^1}^{3\text{个}10^1} + \overbrace{4 \times 10^0}^{4\text{个}10^0}$$

- 从上例可以看出，每一位数都包含了基和权两部分

- ◆ 基：base；权：weight

1234<sub>10</sub>的角标10代表十进制。由于十进制是人类的习惯表达方式，因此在很多时候人们会省略这个角标。

计算机组成与实现

## 数的一般表示方法

- 对于任意一个数，如 $d_{n-1}d_{n-2} \dots d_1d_0$ ，其值表示为

$$d_{n-1} \times B^{n-1} + d_{n-2} \times B^{n-2} + \dots + d_1 \times B^1 + d_0 \times B^0$$

- ◆ 其中 $d_{n-1}$ ， $d_{n-2}$ ， $\dots d_1$ ， $d_0$ 是该进制的可能取值
- ◆ 例如十进制，可能取值为{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

计算机组成与实现

## 二进制的表示方法

- 二进制数的每位也被称为比特（bit）
- 二进制的基能够表示的数字范围只有2个数，即{0,1}
- N位二进制数的各位权从最低位到最高位分别为 $2^0$ ,  $2^1$ , ...,  $2^{N-1}$
- 示例：11011<sub>2</sub>

$$11011_2 = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 16 + 8 + 0 + 2 + 1 = 27_{10}$$

计算机组成与实现

## 常见数制

- 十进制
  - ◆ 各位可能取值：0, 1, 2, 3, 4, 5, 6, 7, 8, 9
  - ◆ 示例：9472<sub>10</sub> = 9472
- 二进制
  - ◆ 各位可能取值：0, 1
  - ◆ 示例：101011<sub>2</sub> = 0b101011
- 十六进制
  - ◆ 各位可能取值：0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
  - ◆ 示例：2A5D<sub>16</sub> = 0x2A5D

十进制	二进制	十六进制
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

计算机组成与实现



## 不同数制的示例

□ 示例

$$\begin{aligned} 9472_{10} &= 9000 + 400 + 70 + 2 \\ &= 9 \times 1000 + 4 \times 100 + 7 \times 10 + 2 \times 1 \\ &= 9 \times 10^3 + 4 \times 10^2 + 7 \times 10^1 + 2 \times 10^0 \\ 9472_{10} &= 2 \times 16^3 + 5 \times 16^2 + 0 \times 16^1 + 0 \times 16^0 \\ &= 2500_{16} \\ 0xA15 &= 0b\ 1010\ 0001\ 0101 \end{aligned}$$

计算机组成与实现

## 进制转换<sup>1/3</sup>

□ 十进制转二进制

- ◆ 用十进制数除以2，得到的余数就是相应二进制的最低位
- ◆ 将得到的商继续除以2，得到的余数就是次低位
- ◆ 重复上述过程直至商为0

□ 示例：27的二进制计算过程

- ◆  $27_{10} = 11011_2$

步骤	被除数	商	余数	备注
1	27	13	1	最低位
2	13	6	1	
3	6	3	0	
4	3	1	1	
5	1	0	1	最高位

计算机组成与实现

## 进制转换<sup>2/3</sup>

### □ 十六进制与二进制

- ◆ 两者之间转换非常简单，基本方法是：每4位二进制对应1位十六进制
- ◆ 示例：110101101011011<sub>2</sub>与6C5B<sub>16</sub>之间的转换

二进制	110	1100	0101	1011
十六进制	6	C	5	B

手工转换时，注意从低位开始转换，以防止高位不足4位导致出错。

计算机组成与实现

## 进制转换<sup>3/3</sup>

### □ 十进制转十六进制

- ◆ 先将十进制数转换为二进制数
- ◆ 再二进制数转换为十六进制数

计算机组成与实现



## 目录

- 常见进制及其转换
- 常见术语
- 二进制加法
- 整数的二进制表示方法
- 浮点数的二进制表示方法
- 补码的几种常见运算

## 字节、字

- 字节（byte）：由8个二进制位构成的一个元组，是目前计算机数据单位
  - ◆ 1个byte的表示范围是从0x00至0xFF，即总共是256个数字
- 字（word）：一个字包含的二进制位数因计算机不同而不同
  - ◆ 不同CPU计算能力不同，因此其字的大小也不同
  - ◆ 以MIPS为例，CPU能计算的数据大小为32位，故其字长为32位
  - ◆ 大量当代计算机，如PC、服务器甚至手机，CPU字长已经发展到64位了
  - ◆ 在一些专用领域，CPU的字长甚至达到128位
  - ◆ 嵌入式领域中的某些CPU字长可能只有16位甚至8位

## 最高/最低有效位、最高/最低有效字节

- MSB（Most Significant Bit）：最高有效位，位于最左位
- LSB（Low Significant Bit）：最低有效位，位于最右边位
- MSB（Most Significant Byte）：最高有效字节，位于数据的最左边的字节
- LSB（Least Significant Byte）：最低有效自己，位于数据的最右边的字节

<u>01011011</u>		<u>BCDEF789</u>	
最高位	最低位	最高字节	最低字节

计算机组成与实现

## 目录

- 常见进制及其转换
- 常见术语
- **二进制加法**
- 整数的二进制表示方法
- 浮点数的二进制表示方法
- 补码的几种常见运算



北京航空航天大学计算机学院  
School of Computer Science and Engineering, Beihang University

## 二进制加法

### □ 回顾十进制数加法计算原理

- ◆ 将两个对应的数字位与来自低位的进位相加，如果其和大于等于10，那么产生进位
- ◆ 计算过程从最低位开始依次向最高位。其中最低位的进位为0

### □ 二进制加法与十进制加法原理类似

- ◆ 二进制每位数字非0即1，因此两个数字位相加最大值为2，即 $10_2$
- ◆ 显然，此时就应该进位了

### □ 示例： $1011_2$ 与 $0011_2$ 的计算过程

	b3	b2	b1	b0
	1	0	1	1
	0	0	1	1
+				1
	1	1	1	0

1)b0-b3位计算(产生进位输出)

	b4	b3	b2	b1	b0
		1	1	1	1
		0	0	0	1
+	1	1	1	1	
	1	0	0	0	0

2)b4位(进位)

计算机组成与实现

## 溢出

### □ 人：在上例中，计算结果为 $10000_2$ 是正常的

- ◆ 人在计算中，通常不考虑位数限制，可以灵活的添加高位

### □ 计算机：由于受硬件限制，就必须考虑计算结果的位数

- ◆ 假设CPU的字长为4位，则b4位是不存在的，故计算结果就是 $0000_2$ ，即0。
- ◆ 这种情况就是溢出。产生溢出，意味着计算结果出现错误了
  - CPU在溢出发生时，会通过称为异常的机制来报告这个错误

	b4	b3	b2	b1	b0
		1	1	1	1
		0	0	0	1
+	1	1	1	1	
	1	0	0	0	0

溢出~Overflow; 异常~Exception

计算机组成与实现

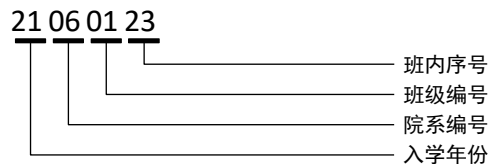


## 目录

- 常见进制及其转换
- 常见术语
- 二进制加法
- 整数的二进制表示方法
- 浮点数的二进制表示方法
- 补码的几种常见运算

## 一切皆可用数字表示

- 误区：很多时候，我们认为数字的含义只能是数值
- 示例1：21060123
  - ◆ 解读1：如果在银行账户中，那就是存款
  - ◆ 解读2：如果在学籍系统中，那就是学号



- 示例2：0和1
  - ◆ 0：可以对应逻辑真（F，false）
  - ◆ 1：可以对应逻辑真（T，true）

## 一切皆可用数字表示

- 启示1:
  - ◆ 一个数字首先是一个**编码**，其**含义**必须结合上下文来解读
- 启示2:
  - ◆ 一个数字可以作为整体解读，也可以分为若干部分解读
- 对于一个数字来说，其包含2部分内容
  - ◆ 编码：就是数字本身，或者说是一种记号。例如007
  - ◆ 语义：编码所代表的概念的含义，是对编码的解释
    - 例如007可以代表邦德，也可以是房间编号

编码本身没有任何意义  
编码只有被赋予语义后才有意义

计算机组成与实现

## 数的编码空间

- 示例：**3位十进制**
  - ◆ 编码空间：000, 001, ..., 999
  - ◆ 编码个数： **$10^3$** ，即1000
- 编码空间：有效编码的集合
- 空间大小：有效编码的总数
  - ◆ 对于一个 $n$ 位 $B$ 进制数来说，其编码空间大小为 $B^n$
  - ◆ 例如
    - 3位二进制，其编码空间大小为 $2^3=8$
    - 3位十六进制，其编码空间为 $16^3=4096$

计算机组成与实现

## 编码位数

- Q: 对于某个B进制来说, 如何确定某个编码方案的编码位数?
- A:
  - ◆ 1) 首先需要确定需要编码的对象数量N
  - ◆ 2) 然后根据语义数量计算位数 $\log_B^N$  (计算结果向上取整)
- 示例: 编码26个英文字母, 需要几位二进制编码?
  - ◆ 1) 需编码的对象总共为56 (26个大写字母, 26个小写字母)
  - ◆ 2) 计算 $\log_2^{56} \approx 5.8$ 。为此至少需要6位二进制

编码与编码对象之间的对应关系是由人确定的  
例如: A可以用000000<sub>2</sub>对应, 也可以用111000<sub>2</sub>对应

计算机组成与实现

## 二进制无符号数

- 二进制无符号数的编码方案

- ◆ 没有负数; 全是自然数

编码 空间	0000 0000 0000 0000 0000 0000 0000 0000 <sub>2</sub> = 0 <sub>10</sub>	} 整数 空间
	0000 0000 0000 0000 0000 0000 0000 0001 <sub>2</sub> = 1 <sub>10</sub>	
	0000 0000 0000 0000 0000 0000 0000 0010 <sub>2</sub> = 2 <sub>10</sub>	
	...	
	0111 1111 1111 1111 1111 1111 1111 1101 <sub>2</sub> = 2,147,483,645 <sub>10</sub>	
	0111 1111 1111 1111 1111 1111 1111 1110 <sub>2</sub> = 2,147,483,646 <sub>10</sub>	
	0111 1111 1111 1111 1111 1111 1111 1111 <sub>2</sub> = 2,147,483,647 <sub>10</sub>	
	1000 0000 0000 0000 0000 0000 0000 0000 <sub>2</sub> = 2,147,483,648 <sub>10</sub>	
	1000 0000 0000 0000 0000 0000 0000 0001 <sub>2</sub> = 2,147,483,649 <sub>10</sub>	
	1000 0000 0000 0000 0000 0000 0000 0010 <sub>2</sub> = 2,147,483,650 <sub>10</sub>	
	...	
	1111 1111 1111 1111 1111 1111 1111 1101 <sub>2</sub> = 4,294,967,293 <sub>10</sub>	
	1111 1111 1111 1111 1111 1111 1111 1110 <sub>2</sub> = 4,294,967,294 <sub>10</sub>	
	1111 1111 1111 1111 1111 1111 1111 1111 <sub>2</sub> = 4,294,967,295 <sub>10</sub>	

无符号~unsigned

计算机组成与实现



## 二进制符号数

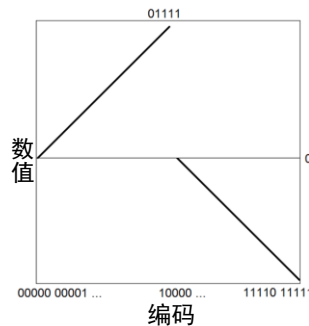
### 原码方案

- ◆ 用最高位表示符号位（0～正，1～负）
- ◆ 其余位代表绝对值（也就是无符号数）

### 示例：4位原码编码方案

### 原码方案的缺陷

- ◆ 存在2个零，即 $0000_2$ 和 $1000_2$ 
  - 1) 有效编码被浪费了
  - 2) 硬件无法使用前述加法
- ◆ 编码数值不连续
  - 希望数值随编码增大而增大



$$0000_2 = 0_{10}$$

$$0001_2 = 1_{10}$$

$$0010_2 = 2_{10}$$

$$0011_2 = 3_{10}$$

$$0100_2 = 4_{10}$$

$$0101_2 = 5_{10}$$

$$0110_2 = 6_{10}$$

$$0111_2 = 7_{10}$$

$$1000_2 = -0_{10}$$

$$1001_2 = -1_{10}$$

$$1010_2 = -2_{10}$$

$$1011_2 = -3_{10}$$

$$1100_2 = -4_{10}$$

$$1101_2 = -5_{10}$$

$$1110_2 = -6_{10}$$

$$1111_2 = -7_{10}$$

符号～signed

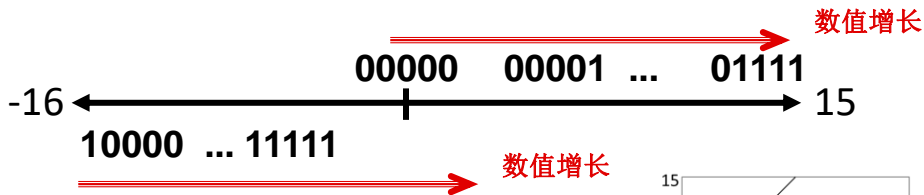
5位原码方案

计算机组成与实现

## 二进制符号数

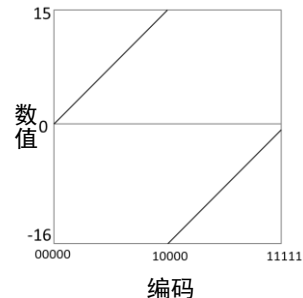
### 二进制补码方案（以5位二进制补码为例）

- ◆ 最高位代表符号位：0～正，1～负
- ◆ 10000代表最小的负数（-16），11111代表最大的负数（-1）



### 特点：

- ◆ 只有1个零
- ◆ 符号确定后，数值随编码增长而增长



5位补码方案

## 32位二进制补码

- 表示范围：-2,147,483,648至2,147,483,647

符号位

```

0000 0000 0000 0000 0000 0000 0000 0000two = 010
0000 0000 0000 0000 0000 0000 0000 0001two = 110
0000 0000 0000 0000 0000 0000 0000 0010two = 210
...
0111 1111 1111 1111 1111 1111 1111 1101two = 2,147,483,64510
0111 1111 1111 1111 1111 1111 1111 1110two = 2,147,483,64610
0111 1111 1111 1111 1111 1111 1111 1111two = 2,147,483,64710
1000 0000 0000 0000 0000 0000 0000 0000two = -2,147,483,64810
1000 0000 0000 0000 0000 0000 0000 0001two = -2,147,483,64710
1000 0000 0000 0000 0000 0000 0000 0010two = -2,147,483,64610
...
1111 1111 1111 1111 1111 1111 1111 1101two = -310
1111 1111 1111 1111 1111 1111 1111 1110two = -210
1111 1111 1111 1111 1111 1111 1111 1111two = -110

```

计算机组成与实现

## 二进制补码的一般性表示

- 对于一个 $x_{N-1}x_{N-2} \dots x_1x_0$ 的 $N$ 位二进制补码数，其值 $A$ 为：

$$\begin{aligned}
 A &= (x_{N-1} \times -2^{N-1}) + (x_{N-2} \times 2^{N-2}) + \dots + (x_1 \times 2^1) + (x_0 \times 2^0) \\
 &= x_{N-1} \times -2^{N-1} + \sum_{i=0}^{N-2} x_i \times 2^i
 \end{aligned}$$

计算机组成与实现

## 二进制补码小结

- 现代计算机普遍采用
- 使用最高位作为符号位
- 正数区间与负数区间大致相同
  - ◆ 负数区间比正数区间多一个数
- 相反数计算方法：各位取反，然后加1
  - ◆ 示例：已知 $+7=0111_2$ ，则 $-7=1000_2+1_2=1001_2$

计算机组成与实现

## 课堂练习

- 假设CPU字长为4位。以下哪个范围可以用二进制补码表示？
  - a) -15 至 +15 空间大小为31，需要至少5位
  - b) 0 至+15 空间大小为16（合理），但没有负数区间
  - c) -8 至 +7 正负区间都有，负区间多1个，空间大小16
  - d) -16 至 +15 正负区间都有，负区间多1个，但空间大小为32

计算机组成与实现



## 目录

- 常见进制及其转换
- 常见术语
- 二进制加法
- 整数的二进制表示方法
- 浮点数的二进制表示方法
- 补码的几种常见运算

## 浮点数概述

- 计算机除了处理整数外，很多时候也需要处理浮点数
  - ◆ 例如圆周率3.1415926，就无法用前面讲的二进制整数编码方案表示
  - ◆ 在工程计算中，就大量涉及浮点数
- 浮点数具有表示范围和精度都较高的特点
  - ◆ 它解决了整数和小数位长度固定的限制
  - ◆ 允许表示一个很大的数或者很小的数



William M. Kahan (威廉·凯亨)，1933年6月  
领导开发了Intel的8087浮点协处理器  
制定了IEEE754和854标准  
浮点数标准之父；1989年图灵奖获得者  
*For his fundamental contributions to numerical  
analysis. One of the foremost experts on floating-  
point computations. Kahan has dedicated himself to  
"making the world safe for numerical computations"!*

# 浮点数格式

- 科学计数法回顾：5120可以表示 $5.23 \times 10^3$ ，包含3部分
  - 尾数（mantissa, M）：5.23
  - 基数（base, B）：10
  - 指数（exponent, E）：3
- 浮点数的方法与科学记数法非常相似，一个浮点数可以表示为：  

$$\pm M \times B^E$$

- 4部分：符号（sign, S）、尾数、基数、指数

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	指数								尾数																						

- 符号：1位，0~正，1~负
- 指数：8位，用于表示范围
- 尾数：23位，用于表示精度
- 基数：由于基数固定为2，因此就被省略了

指数和尾数，增加任何一位数都会减少另一方位数  
 目前方案是在精度与表示范围之间反复权衡后的结果

计算机组成与实现

# 浮点数格式（优化前）

- 示例：用二进制浮点数格式表示 $5.23 \times 10^3$ 
  - $5.23 \times 10^3 = 5230_{10} = 1010001101110 = 1.01000110111 \times 2^{12}$
  - 符号为0，指数为12（1100<sub>2</sub>），尾数为1.01000110111

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	指数								尾数																						
0	0	0	0	0	1	1	0	0	1	0	1	0	0	0	1	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0

计算机组成与实现

## 浮点数格式（优化后）

- 科学记数法通常不会让0出现在尾数的第1位（即小数点左边那位）。类似，浮点数表示方法同样不会让0出现在尾数的第1位
  - 由于尾数的第1位只能为1，那么就没有必要再占用实际存储位
  - 这使尾数多了1位有效存储位
  - 在计算过程中还需要将该位自动补回
- 格式优化：5.23 × 10<sup>3</sup>的尾数1.01000110111，优化为01000110111

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	指数								尾数																						
0	0	0	0	0	1	1	0	0	1	0	1	0	0	0	1	1	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0

优化前的浮点数格式

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	指数								尾数																						
0	0	0	0	0	1	1	0	0	0	1	0	0	0	1	1	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0

优化后的浮点数格式

计算机组成与实现

## 表示绝对值小于0的浮点数

- 绝对值小于0的浮点数，其指数是负的
  - 示例：0.5和0.25，其对应的就是 $1.0 \times 2^{-1}$ 与 $1.0 \times 2^{-2}$
- 思路：指数采用二进制补码
- 缺点：负指数的浮点数虽较小，但其二进制却似乎是个较大的数
  - 示例：0.5与2.0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	指数								尾数																						
0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

0.5 ( $1.0 \times 2^{-1}$ ) 的浮点数格式为：0x7F800000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	指数								尾数																						
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

2.0 ( $1.0 \times 2^1$ ) 的浮点数格式：0x00800000

浮点数：0.5 < 2.0

二进制：0x7F800000 > 0x00800000

计算机组成与实现

### 表示绝对值小于0的浮点数：偏阶计数法

- 目标：00000000<sub>2</sub>对应最小的负指数，而11111111<sub>2</sub>对应最大的正指数
- 思路：编码用的指数=真实指数+127；该方法被称为偏阶记数法
- 示例：0.5与2.0

- 0.5的偏阶编码指数：11111111<sub>2</sub>+01111111<sub>2</sub>=01111110<sub>2</sub>
- 2.0的偏阶编码指数：00000001<sub>2</sub>+01111111<sub>2</sub>=10000000<sub>2</sub>

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0					
S	指数								尾数																											
0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0					

0.5 (1.0×2<sup>-1</sup>) 的偏阶浮点数格式为：0x3F000000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0					
S	指数								尾数																											
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0					

2.0 (1.0×2<sup>+1</sup>) 的偏阶浮点数格式为：0x40000000

IEEE754浮点标准就采用了偏阶记数法 计算机组成与实现

### 单精度浮点表示范围

- 单精度浮点：占据32位
  - 最大正数：+3.402824 × 10<sup>38</sup>
    - 符号位：0，正数
    - 指数：127（编码指数为254） 最大正数计算方法
    - 尾数：11111111111111111111111111111111（23个1）
$$-1^0 \times 1.M \times 2^E = +1.11111111111111111111111111111111 \times 2^{127} \approx 3.402823 \times 10^{38}$$
  - 最小正数：+1.175494 × 10<sup>-38</sup>
  - 最大负数：-1.175494 × 10<sup>-38</sup>
  - 最小负数：-3.402824 × 10<sup>38</sup>
- 单精度表示区间
  - 正数区间：+1.175494 × 10<sup>-38</sup>至+3.402824 × 10<sup>38</sup>
  - 负数区间：-3.402824 × 10<sup>38</sup>至-1.175494 × 10<sup>-38</sup>

计算机组成与实现

## 双精度浮点表示范围

- 为了提供更大的取值范围和更高的精度，IEEE754标准还定义了双精度浮点
- 双精度浮点：占据64位

格式	总位数	符号位	指数位	尾数位
单精度浮点	32	1	8	23
双精度浮点	64	1	11	52

- 双精度表示范围

- ◆ 正数区间： $+2.22507385850720 \times 10^{-308}$ 至 $+1.79769313486232 \times 10^{308}$
- ◆ 负数区间： $-1.79769313486232 \times 10^{308}$ 至 $-2.22507385850720 \times 10^{-308}$

计算机组成与实现

## 特殊情况

- IEEE754标准用指数和尾数的某些特殊编码表示一些特殊情况
  - ◆ 例如如：0、 $\pm\infty$ 、非法数（或者不存在的数，如 $\sqrt{-1}$ ）

表示的数	单精度浮点数			双精度浮点数		
	符号	指数	尾数	符号	指数	尾数
0	X	0	0	X	0	0
$+\infty$	0	255	0	0	2047	0
$-\infty$	1	255	0	1	2047	0
NaN（非数）	X	255	非0	X	2047	非0
正的浮点数	0	1~254	任意	0	1~2046	任意
负的浮点数	1	1~254	任意	1	1~2046	任意

注意：IEEE754编码方案中有2个0

计算机组成与实现



## 上溢和下溢

- 以单精度浮点数为例，其区间为  
 $[-3.402824 \times 10^{38}, -1.175494 \times 10^{-38}]$ ,  $[+1.175494 \times 10^{-38}, +3.402824 \times 10^{38}]$
- 当结果A:  $A > 3.402824 \times 10^{38}$  或  $A < -3.402824 \times 10^{38}$ 
  - ◆ 这种情况被称为上溢
  - ◆ A被向上舍入为 $\pm\infty$
- 当结果A:  $-1.175494 \times 10^{-38} < A < 0$  或  $0 < A < +1.175494 \times 10^{-38}$ 
  - ◆ 这种情况被称为下溢
  - ◆ A被向下舍入为0

计算机组成与实现



北京航空航天大学计算机学院  
School of Computer Science and Engineering, Beihang University

## 目录

- 常见进制及其转换
- 常见术语
- 二进制加法
- 整数的二进制表示方法
- 浮点数的二进制表示方法
- 补码的几种常见运算

## 负数的二进制补码的计算方法

- 相反数：符号相反但绝对值相同的一对整数
  - ◆ 假设a与b是2个整数，如果a+b=0，则a与b是相反数
- 显然a与-a就是相反数
  - ◆ 示例：+7与-7是相反数，-100与+100是相反数
- 相反数计算方法
  - ◆ 方法：X的二进制补码表示为 $x_{N-1} \dots x_1 x_0$ ，则-X为X的各位取反然后加1
  - ◆ 原理：
    - 由于 $x_{N-1} \dots x_1 x_0$ 与 $\bar{x}_{N-1} \dots \bar{x}_1 \bar{x}_0$ 各对应位的2位为0、1互斥，因此相加后N位计算结果必然是全1，即：
 
$$x_{N-1} \dots x_1 x_0 + \bar{x}_{N-1} \dots \bar{x}_1 \bar{x}_0 = 1 \dots 11$$
    - 二进制补码中的全1代表-1，故： $x + \bar{x} = -1$
    - 将上式变形，得： $-x = \bar{x} + 1$

计算机组成与实现

## 负数的二进制补码的计算方法

- 示例1：用8位二进制补码表示-14
  - ◆ 1) 14的二进制补码表示为00001110<sub>2</sub>
  - ◆ 2) 将该数各位取反可得11110001<sub>2</sub>
  - ◆ 3) 再计算11110001<sub>2</sub>+1=11110010<sub>2</sub>
  - ◆ -14的8位二进制补码为11110010<sub>2</sub>
- 示例2：-14的二进制补码为11110010<sub>2</sub>，计算14的二进制补码
  - ◆ 1) 将11110010<sub>2</sub>各位取反可得00001101<sub>2</sub>
  - ◆ 2) 再计算00001101<sub>2</sub>+1=00001110<sub>2</sub>
  - ◆ 14的8位二进制补码为00001110<sub>2</sub>

计算机组成与实现

## 减法

- 思路：为了计算 $x-y$ ，可以通过数学变形改为计算 $x+(-y)$
- 示例：计算 $28-14$ （假设字长为8位）
  - ◆ 1) 28的8位二进制补码为：00011100<sub>2</sub>
  - ◆ 2) -14的二进制补码为：
    - 14的二进制补码为：00001110<sub>2</sub>
    - 各位取反为：11110001<sub>2</sub>
    - 加1，得：11110010<sub>2</sub>
  - ◆ 3) 计算：00011100<sub>2</sub>+11110010<sub>2</sub>=00001110<sub>2</sub>=14

由于字长仅有8位，  
因此b8位被舍弃。  
注意：b8位本质上是属于溢出了

$$\begin{array}{cccccccc}
 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\
 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\
 + & 1 & 1 & 1 & & & & & \\
 \hline
 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0
 \end{array}$$

补码体系的优势：只需要设计加法运算的硬件，减法可以转换为加法运算，故不需为减法设计相应硬件

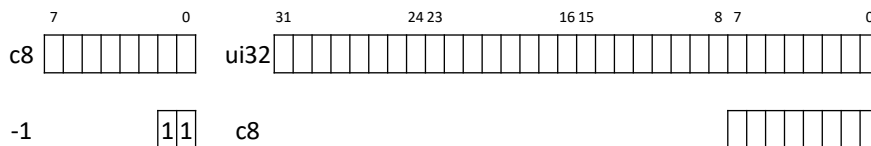
## 位扩展

- ❑ 程序中存在不同类型间的变量赋值

```
1 char          c8 ; —— 8位符号数
2 int           i32 ; —— 32位符号数
3 unsigned int  ui32 ; —— 32位无符号数
4
5 c8            = -1 ;
6 i32           = c8 ;
7 ui32          = (unsigned char)c8 ;
```

- 不同类型的变量往往位数不同，这就涉及扩展问题

- Line5: -1最少用2位二进制补码表示，即11<sub>2</sub>，但c8需要8位
- Line7: ui32是32位，c8是8位

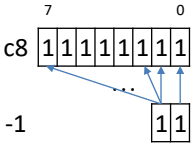


计算机组成与实现

# 位扩展

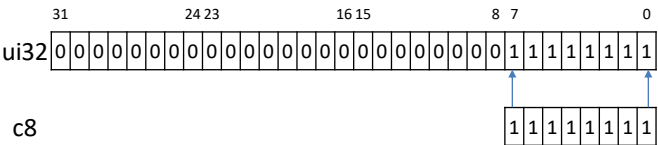
□ 符号扩展：对于Line5来说，由于c8是符号数，因此需要将-1的符号位复制到c8的高位部分。

◆ 计算完成后， $c8=11111111_2$ ，即8位补码的-1



□ 无符号扩展：对于Line7来说，由于ui32是无符号数，因此需要将c8视为无符号数，并且为确保扩展后的结果正确，ui32的高位部分应该都为0。

◆ 计算完成后， $ui32=00000000000000000000000011111111_2$



计算机组成与实现

# 比较

□ C语言定义了多个比较操作，其运算结果是逻辑的真假值

◆ =、≠、>、<、≥、≤

□ 比较运算的等价变换：实现< 和=这两种比较运算，再结合NOT逻辑运算，就能实现上述六种比较运算

原运算	$A \neq B$	$A > B$	$A \geq B$	$A \leq B$
等价运算	$\overline{A = B}$	$B < A$	$\overline{A < B}$	$\overline{B < A}$

计算机组成与实现

### 比较：小于运算(符号数)<sup>1/3</sup>

- 如果A和B是符号数： $A < B$ 小于运算可以转换为 $A - B < 0$ 
  - ◆ 1) 首先执行减法，即 $C \leftarrow A - B$
  - ◆ 2) 判断C的符号位：
    - 如果为1，则 $C < 0$ ，即 $A < B$ 为真
    - 如果为0，则 $C \geq 0$ ，即 $A < B$ 为假
- 示例：A、B为4位二进制补码的 $0000_2$ 与 $1000_2$ ，计算“ $A < B$ ”的真假值
  - ◆ 理论分析： $0000_2$ 和 $1000_2$ 分别为0和-8，显然 $0000_2$ 大于 $1000_2$ ，故“ $A < B$ ”为假
  - ◆ 实际计算： $A - B = A + \bar{B} + 1 = 0000 + 0111 + 0001 = 1000$
  - ◆ 计算的结论：结果的符号位b3为1，即 $A - B < 0$ ，意味着“ $A < B$ ”为真

计算机组成与实现

### 比较：小于运算(符号数)<sup>2/3</sup>

- 正确的理论计算
  - ◆ 从数学上可知，0减-8的结果应该是+8
  - ◆ 如果用补码表示+8，则至少需要5位二进制补码，即 $01000_2$
  - ◆ 由于b4的存在，故b3的“1”含义是数值，而不是符号
- 错误的实际计算
  - ◆ 前述计算采用4位二进制补码计算，结果为 $1000_2$
  - ◆ 由于b4的缺失，故b3的“1”被错误的解读为符号
- 错误的原因：本质上是由于位数不足，导致计算结果+8超出了4位二进制补码的表示范围，发生了溢出

为了完成减法，必须从b4借位

## 比较：小于运算(符号数)<sup>3/3</sup>

- 解决方案：扩展符号位后再计算
  - ◆ 2个N位二进制补码的操作数均扩展1位符号位
  - ◆ 进行N+1位二进制补码的减法计算
- 示例：A、B为4位二进制补码的0000<sub>2</sub>与1000<sub>2</sub>，计算“A<B”的真假值
  - ◆ 1) 将A、B符号位扩展1位，分别为00000<sub>2</sub>和10000<sub>2</sub>
  - ◆ 2) 计算： $A - B = A + \bar{B} + 1 = 00000 + 00111 + 00001 = 01000$
  - ◆ 3) 根据符号位b4为0可知 $A - B \geq 0$ ，故“A<B”为假
  - ◆ 与理论分析一致！

为了完成减法，必须从b4借位

## 比较：小于运算(无符号数)

- 解决方案：0扩展后再按符号数计算
  - ◆ 1) 如果比较的两个数均为无符号数，将其视为符号数
  - ◆ 2) 为了能够用符号数正确表达其原值，需要将两个数分别扩展1位符号位，且符号位为0
  - ◆ 3) 之后，就可以采用前述的方案了

计算机组成与实现

## 相等

- 方案1：采用XOR运算，然后判断结果是否全0
  - ◆ 1) 执行XOR运算，即  $C \leftarrow A \oplus B$
  - ◆ 2) 将C的各位OR起来，得到1位结果
    - 如果为0，即“A=B”为真
    - 如果为1，即“A=B”为假
- 方案2：采用减法运算，然后判断结果是否全0
  - ◆ 1) 执行减法运算，即  $C \leftarrow A - B$
  - ◆ 2) 将C的各位OR起来，得到1位结果
    - 如果为0，即“A=B”为真
    - 如果为1，即“A=B”为假

计算机组成与实现

## 乘法：无符号数

- 二进制乘法的基本计算过程与十进制乘法基本类似
- 示例：计算  $0101_2 \times 1011_2$

- ◆ 理论分析结果：  $0101_2 \times 1011_2 = 5 \times 11 = 55$
- ◆ 实际计算结果：  $0110111_2 = 55$

$$\begin{array}{r}
 0101 \\
 \times 1011 \\
 \hline
 0101 \\
 0100 \\
 0000 \\
 + 0101 \\
 \hline
 0110111
 \end{array}$$

- 基本原理：循环累加左移后的被乘数
  - ◆ 假设被乘数A为N位，乘数B为M位
  - ◆ 结果C为N+M-1位

```

C = 0
for i = 0 to M-1
  if B[i] then
    C = C + A<<i
  
```

计算机组成与实现

## 乘法：符号数

### □ 基本原理：借助无符号数乘法

- ◆ 假设A、B均为32位符号数，C为64位计算结果
- ◆ 1) 得到A、B的绝对值A'和B'
  - 由于A和B均为符号数，故A'和B'均为31位
- ◆ 2) 借助无符号乘法，计算乘法结果的绝对值 $C' = A' \times B'$ 
  - 从计算的角度，C'的位数为61位（ $31+31-1=61$ ）
- ◆ 3) 根据A和B的符号位决定结果的正或负
  - 同号为正，异号为负
- ◆ 4) 根据结果计算C
  - 如果结果为正： $C = 000||C'$
  - 如果结果为负： $C = \overline{000||C'} + 1$

A[31]	B[31]	结果
0	0	正
0	1	负
1	0	负
1	1	正

计算机组成与实现

## 除法的复杂性

### □ 运算结果种类

- ◆ 乘法：运算结果只有一个，即积
- ◆ 除法：有2个运算结果，即商和余数

### □ 结果正负性质

- ◆ 数学公式：被除数 = 除数  $\times$  商 + 余数
- ◆ 示例： $-7 \div 2$ 。如果仅按数学公式，可以有2种结果
  - 结果1： $-7 = 2 \times (-3) + (-1)$
  - 结果2： $-7 = 2 \times (-4) + 1$

### □ 中间计算过程

- ◆ 乘法：以加法为核心
- ◆ 除法：以减法为核心。减法比加法要复杂些。

计算机组成与实现



## 除法：无符号数<sup>1/3</sup>

- 示例：计算 $1001010_2$ 除以 $1000_2$ 
  - ◆ 理论分析： $1001010_2$ 为74， $1000_2$ 为8，故 $74 \div 8$ 的商为9，余数为2
  - ◆ 计算过程：与十进制除法类似，核心是“试商”
- 试商：在二进制除法中，商的各位非0即1，“试商”更为简单
  - ◆ 计算商的每位时，仅需做一次减法
  - ◆ 然后根据减法结果的符号位判断是否够减

位序	6 5 4 3 2 1 0	
	1 0 0 1	商
除数 1 0 0 0	1 0 0 1 0 1 0	被除数
	- 1 0 0 0	
	1 0 1 0	
	- 1 0 0 0	
	0 0 1 0	余数

### 注意

在示例的计算过程中，b<sub>2</sub>、b<sub>1</sub>的试商过程被省略了。计算机是很“傻”的，因此在实际计算中必须给出完整的计算过程。

计算机组成与实现

## 除法：无符号数<sup>2/3</sup>

- 中间余数：假设除数为N位，则中间余数为N+1位
- 计算方法：每一次迭代计算商的一位
  - ◆ 每次从被除数中取出1位，构成中间余数的末位
  - ◆ 试商：用中间余数减除数，根据减法结果决定商的各位取值

位序	6 5 4 3 2 1 0	
	0 0 0 1 0 0 1	商
除数 1 0 0 0	0 0 0 0 1 0 0 1 0 1 0	被除数。初始中间余数为00001 <sub>2</sub> 。
	- 0 1 0 0 0	
	0 0 0 1 0	差值<0：商位6为0；中间余数=中间余数左移1位    被除数位5
	- 0 1 0 0 0	差值<0：商位5为0；中间余数=中间余数左移1位    被除数位4
	0 0 1 0 0	差值<0：商位4为0；中间余数=中间余数左移1位    被除数位3
	- 0 1 0 0 0	差值<0：商位3为1；中间余数=差值左移1位    被除数位2
	0 0 0 1 0	差值<0：商位2为0；中间余数=中间余数左移1位    被除数位1
	- 0 1 0 0 0	差值<0：商位1为0；中间余数=中间余数左移1位    被除数位0
	0 1 0 1 0	差值≥0：商位0为1；计算到最后一位，停止计算；中间余数=差值
	- 0 1 0 0 0	
	0 0 0 1 0	

## 除法：符号数<sup>3/3</sup>

### □ 符号数除法的基本思路

- ◆ 1) 获得被除数、除数的绝对值
- ◆ 2) 执行两者绝对值的除法，得到商和余数
- ◆ 3) 商的符号：由被除数与除数的符号决定，即同号为正、异号为负
- ◆ 4) 余数的符号：与被除数的符号相同

计算机组成与实现